pytest-mock

Bruno Oliveira

May 16, 2023

CONTENTS:

This pytest plugin provides a `mocker` fixture which is a thin-wrapper around the patching API provided by the mock package:

```
import os

class UnixFS:

    @staticmethod
    def rm(filename):
        os.remove(filename)

def test_unix_fs(mocker):
    mocker.patch('os.remove')
    UnixFS.rm('file')
    os.remove.assert_called_once_with('file')
```

Besides undoing the mocking automatically after the end of the test, it also provides other nice utilities such as `spy` and `stub`, and uses pytest introspection when comparing calls.

CHAPTER

ONE

INSTALL

Install using pip:

```
$ pip install pytest-mock
```

1.1 Usage

The `mocker` fixture has the same API as mock.patch, supporting the same arguments:

```
def test_foo(mocker):
    # all valid calls
    mocker.patch('os.remove')
    mocker.patch.object(os, 'listdir', autospec=True)
    mocked_isfile = mocker.patch('os.path.isfile')
```

The supported methods are:

- mocker.patch
- mocker.patch.object
- mocker.patch.multiple
- mocker.patch.dict
- mocker.stopall
- mocker.stop
- `mocker.resetall()`: calls reset_mock() in all mocked objects up to this point.

Also, as a convenience, these names from the `mock` module are accessible directly from `mocker`:

- Mock
- MagicMock
- PropertyMock
- ANY
- DEFAULT
- call
- sentinel
- mock_open

- seal

It is also possible to use mocking functionality from fixtures of other scopes using the appropriate fixture:

- `class_mocker`
- `module_mocker`
- `package_mocker`
- `session_mocker`

1.1.1 Spy

The `mocker.spy` object acts exactly like the original method in all cases, except the spy also tracks function/method calls, return values and exceptions raised.

```
def test_spy_method(mocker):
    class Foo(object):
        def bar(self, v):
            return v * 2

    foo = Foo()
    spy = mocker.spy(foo, 'bar')
    assert foo.bar(21) == 42

    spy.assert_called_once_with(21)
    assert spy.spy_return == 42

def test_spy_function(mocker):
    # mymodule declares `myfunction` which just returns 42
    import mymodule

    spy = mocker.spy(mymodule, "myfunction")
    assert mymodule.myfunction() == 42
    assert spy.call_count == 1
    assert spy.spy_return == 42
```

The object returned by `mocker.spy` is a `MagicMock` object, so all standard checking functions are available (like `assert_called_once_with` or `call_count` in the examples above).

In addition, spy objects contain two extra attributes:

- `spy_return`: contains the returned value of the spied function.
- `spy_exception`: contain the last exception value raised by the spied function/method when it was last called, or `None` if no exception was raised.

Besides functions and normal methods, `mocker.spy` also works for class and static methods.

As of version 3.0.0, `mocker.spy` also works with `async def` functions.

Note: In versions earlier than `2.0`, the attributes were called `return_value` and `side_effect` respectively, but due to incompatibilities with `unittest.mock` they had to be renamed (see #175 for details).

As of version 3.10, spying can be also selectively stopped.

```
def test_with_unspy(mocker):
    class Foo:
        def bar(self):
            return 42

    spy = mocker.spy(Foo, "bar")
    foo = Foo()
    assert foo.bar() == 42
    assert spy.call_count == 1
    mocker.stop(spy)
    assert foo.bar() == 42
    assert spy.call_count == 1
```

`mocker.stop()` can also be used by `mocker.patch` calls.

1.1.2 Stub

The stub is a mock object that accepts any arguments and is useful to test callbacks. It may receive an optional name that is shown in its `repr`, useful for debugging.

```
def test_stub(mocker):
    def foo(on_something):
        on_something('foo', 'bar')

    stub = mocker.stub(name='on_something_stub')

    foo(stub)
    stub.assert_called_once_with('foo', 'bar')
```

See also:

`async_stub` method, which actually the same as `stub` but makes async stub.

1.2 Configuration

1.2.1 Use standalone "mock" package

Python 3 users might want to use a newest version of the `mock` package as published on PyPI than the one that comes with the Python distribution.

```
[pytest]
mock_use_standalone_module = true
```

This will force the plugin to import `mock` instead of the `unittest.mock` module bundled with Python 3.4+.

1.2.2 Improved reporting of mock call assertion errors

This plugin monkeypatches the mock library to improve pytest output for failures of mock call assertions like `Mock.assert_called_with()` by hiding internal traceback entries from the `mock` module.

It also adds introspection information on differing call arguments when calling the helper methods. This features catches *AssertionError* raised in the method, and uses pytest's own advanced assertions to return a better diff:

```
mocker = <pytest_mock.MockerFixture object at 0x0381E2D0>

    def test(mocker):
        m = mocker.Mock()
        m('fo')
>       m.assert_called_once_with('', bar=4)
E       AssertionError: Expected call: mock('', bar=4)
E       Actual call: mock('fo')
E
E       pytest introspection follows:
E
E       Args:
E       assert ('fo',) == ('',)
E         At index 0 diff: 'fo' != ''
E         Use -v to get the full diff
E       Kwargs:
E       assert {} == {'bar': 4}
E         Right contains more items:
E         {'bar': 4}
E         Use -v to get the full diff


test_foo.py:6: AssertionError
========================== 1 failed in 0.03 seconds ===========================
```

This is useful when asserting mock calls with many/nested arguments and trying to quickly see the difference.

This feature is probably safe, but if you encounter any problems it can be disabled in your `pytest.ini` file:

```
[pytest]
mock_traceback_monkeypatch = false
```

Note that this feature is automatically disabled with the `--tb=native` option. The underlying mechanism used to suppress traceback entries from `mock` module does not work with that option anyway plus it generates confusing messages on Python 3.5 due to exception chaining

1.3 Remarks

1.3.1 Type annotations

`pytest-mock` is fully type annotated, letting users use static type checkers to test their code.

The `mocker` fixture returns `pytest_mock.MockerFixture` which can be used to annotate test functions:

```python
from pytest_mock import MockerFixture

def test_foo(mocker: MockerFixture) -> None:
    ...
```

The type annotations have been checked with `mypy`, which is the only type checker supported at the moment; other type-checkers might work but are not currently tested.

Why bother with a plugin?

There are a number of different `patch` usages in the standard `mock` API, but IMHO they don't scale very well when you have more than one or two patches to apply.

It may lead to an excessive nesting of `with` statements, breaking the flow of the test:

```python
import mock

def test_unix_fs():
    with mock.patch('os.remove'):
        UnixFS.rm('file')
        os.remove.assert_called_once_with('file')

        with mock.patch('os.listdir'):
            assert UnixFS.ls('dir') == expected
            # ...

    with mock.patch('shutil.copy'):
        UnixFS.cp('src', 'dst')
        # ...
```

One can use `patch` as a decorator to improve the flow of the test:

```python
@mock.patch('os.remove')
@mock.patch('os.listdir')
@mock.patch('shutil.copy')
def test_unix_fs(mocked_copy, mocked_listdir, mocked_remove):
    UnixFS.rm('file')
    os.remove.assert_called_once_with('file')

    assert UnixFS.ls('dir') == expected
    # ...

    UnixFS.cp('src', 'dst')
    # ...
```

But this poses a few disadvantages:

- test functions must receive the mock objects as parameter, even if you don't plan to access them directly; also, order depends on the order of the decorated `patch` functions;
- receiving the mocks as parameters doesn't mix nicely with pytest's approach of naming fixtures as parameters, or `pytest.mark.parametrize`;
- you can't easily undo the mocking during the test execution;

An alternative is to use `contextlib.ExitStack` to stack the context managers in a single level of indentation to improve the flow of the test:

```
import contextlib
import mock

def test_unix_fs():
    with contextlib.ExitStack() as stack:
        stack.enter_context(mock.patch('os.remove'))
        UnixFS.rm('file')
        os.remove.assert_called_once_with('file')

        stack.enter_context(mock.patch('os.listdir'))
        assert UnixFS.ls('dir') == expected
        # ...

        stack.enter_context(mock.patch('shutil.copy'))
        UnixFS.cp('src', 'dst')
        # ...
```

But this is arguably a little more complex than using `pytest-mock`.

1.3.2 Usage as context manager

Although mocker's API is intentionally the same as `mock.patch`'s, its use as context manager and function decorator is **not** supported through the fixture:

```
def test_context_manager(mocker):
    a = A()
    with mocker.patch.object(a, 'doIt', return_value=True, autospec=True):  # DO NOT DO␣
↪THIS
        assert a.doIt() == True
```

The purpose of this plugin is to make the use of context managers and function decorators for mocking unnecessary, so it will emit a warning when used as such.

If you really intend to mock a context manager, `mocker.patch.context_manager` exists which won't issue the above warning.

1.4 Contributing

Contributions are welcome! After cloning the repository, create a virtual env and install `pytest-mock` in editable mode with `dev` extras:

```
$ pip install --editable .[dev]
$ pre-commit install
```

Tests are run with `tox`, you can run the baseline environments before submitting a PR:

```
$ tox -e py38
```

Style checks and formatting are done automatically during commit courtesy of pre-commit.

1.5 About

1.5.1 Tidelift

If you use this plugin in a corporate environment, consider supporting `pytest-mock` via Tidelift.

1.5.2 License

Distributed under the terms of the MIT license.

1.5.3 Security contact information

To report a security vulnerability, please use the Tidelift security contact. Tidelift will coordinate the fix and disclosure.

1.6 Changelog

hide-toc

1.6.1 Releases

3.10.0 (2022-10-05)

- Added new `mocker.stop(m)` method to stop specific `mocker.patch` or `mocker.spy` calls (#319).

3.9.0 (2022-09-28)

- Expose `NonCallableMagicMock` via the `mocker` fixture (#318).

3.8.2 (2022-07-05)

- Fixed `AsyncMock` support for Python 3.7+ in `mocker.async_stub` (#302).

3.8.1 (2022-06-24)

- Fixed regression caused by an explicit `mock` dependency in the code (#298).

3.8.0 (2022-06-24)

- Add `MockerFixture.async_mock` method. Thanks @PerchunPak for the PR (#296).

3.7.0 (2022-01-28)

- Python 3.10 now officially supported.
- Dropped support for Python 3.6.

3.6.1 (2021-05-06)

- Fix `mocker.resetall()` when using `mocker.spy()` (#237). Thanks @blaxter for the report and @shadycuz for the PR.

3.6.0 (2021-04-24)

- pytest-mock no longer supports Python 3.5.
- Correct type annotations for `mocker.patch.object` to also include the string form. Thanks @plannigan for the PR (#235).
- `reset_all` now supports `return_value` and `side_effect` keyword arguments. Thanks @alex-marty for the PR (#214).

3.5.1 (2021-01-10)

- Use `inspect.getattr_static` instead of resorting to `object.__getattribute__` magic. This should better comply with objects which implement a custom descriptor protocol. Thanks @yesthesoup for the PR (#224).

3.5.0 (2021-01-04)

- Now all patch functions will emit a warning instead of raising a `ValueError` when used as a context-manager. Thanks @iforapsy for the PR (#221).
- Additionally, `mocker.patch.context_manager` is available when the user intends to mock a context manager (for example `threading.Lock` object), which will not emit that warning.

3.4.0 (2020-12-15)

- Add *mock.seal* alias to the *mocker* fixture (#211). Thanks @coiax for the PR.
- Fixed spying on exceptions not covered by the `Exception` superclass (#215), like `KeyboardInterrupt` – PR #216 by @webknjaz.

  Before the fix, both `spy_return` and `spy_exception` were always assigned to `None` whenever such an exception happened. And after this fix, `spy_exception` is set to a correct value of an exception that has actually happened.

3.3.1 (2020-08-24)

- Introduce `MockFixture` as an alias to `MockerFixture`.

  Before `3.3.0`, the fixture class was named `MockFixture`, but was renamed to `MockerFixture` to better match the `mocker` fixture. While not officially part of the API, it was later discovered that this broke the code of some users which already imported `pytest_mock.MockFixture` for type annotations, so we decided to reintroduce the name as an alias.

  Note however that this is just a stop gap measure, and new code should use `MockerFixture` for type annotations.

- Improved typing for `MockerFixture.patch` (#201). Thanks @srittau for the PR.

3.3.0 (2020-08-21)

- `pytest-mock` now includes inline type annotations and exposes them to user programs. The `mocker` fixture returns `pytest_mock.MockerFixture`, which can be used to annotate your tests:

```
from pytest_mock import MockerFixture

def test_foo(mocker: MockerFixture) -> None:
    ...
```

  The type annotations were developed against mypy version `0.782`, the minimum version supported at the moment. If you run into an error that you believe to be incorrect, please open an issue.

  Many thanks to @staticdev for providing the initial patch (#199).

3.2.0 (2020-07-11)

- AsyncMock is now exposed in `mocker` and supports provides assertion introspection similar to `Mock` objects.

  Added by @tirkarthi in #197.

3.1.1 (2020-05-31)

- Fixed performance regression caused by the `ValueError` raised when `mocker` is used as context manager (#191).

3.1.0 (2020-04-18)

- New mocker fixtures added that allow using mocking functionality in other scopes:
  - `class_mocker`
  - `module_mocker`
  - `package_mocker`
  - `session_mocker`

  Added by @scorphus in #182.

3.0.0 (2020-03-31)

- Python 2.7 and 3.4 are no longer supported. Users using `pip 9` or later will install a compatible version automatically.
- `mocker.spy` now also works with `async def` functions (#179). Thanks @frankie567 for the PR!

2.0.0 (2020-01-04)

Breaking Changes

- `mocker.spy` attributes for tracking returned values and raised exceptions of its spied functions are now called `spy_return` and `spy_exception`, instead of reusing the existing `MagicMock` attributes `return_value` and `side_effect`.

  Version `1.13` introduced a serious regression: after a spied function using `mocker.spy` raises an exception, further calls to the spy will not call the spied function, always raising the first exception instead: assigning to `side_effect` causes `unittest.mock` to behave this way (#175).
- The deprecated `mock` alias to the `mocker` fixture has finally been removed.

1.13.0 (2019-12-05)

- The object returned by `mocker.spy` now also tracks any side effect of the spied method/function.

1.12.1 (2019-11-20)

- Fix error if `mocker.patch` is used in code where the source file is not available, for example stale `.pyc` files (#169).

1.12.0 (2019-11-19)

- Now all patch functions also raise a `ValueError` when used as a context-manager. Thanks @AlexGascon for the PR (#168).

1.11.2 (2019-10-19)

- The *pytest introspection follows* message is no longer shown if there is no pytest introspection (#154). Thanks @The-Compiler for the report.
- `mocker` now raises a `ValueError` when used as a context-manager. Thanks @binarymason for the PR (#165).

1.11.1 (2019-10-04)

- Fix `mocker.spy` on Python 2 when used on non-function objects which implement `__call__` (#157). Thanks @pbasista for the report.

1.11.0

- The object returned by `mocker.spy` now also tracks the return value of the spied method/function.

1.10.4

- Fix plugin when 'terminal' plugin is disabled

1.10.3

- Fix test suite in Python 3.8. Thanks @hroncok for the report and @blueyed for the PR (#140).

1.10.2

- Fix bug at the end of the test session when a call to `patch.stopall` is done explicitly by user code. Thanks @craiga for the report (#137).

1.10.1

- Fix broken links and update README. Also the code is now formatted using black.

1.10.0

- Add support for the recently added `assert_called` method in Python 3.6 and `mock-2.0`. Thanks @rouge8 for the PR (#115).

1.9.0

- Add support for the recently added `assert_called_once` method in Python 3.6 and `mock-2.0`. Thanks @rouge8 for the PR (#113).

1.8.0

- Add aliases for `NonCallableMock` and `create_autospec` to `mocker`. Thanks @mlhamel for the PR (#111).

1.7.1

- Fix `setup.py` to correctly read the `README.rst`. Thanks @ghisvail for the fix (#107).

1.7.0

Incompatible change

- `pytest-mock` no longer supports Python 2.6 and Python 3.3, following the lead of `pytest` and other projects in the community. Thanks @hugovk for the PR (#96).

Packaging

- Fix `mock` requirement in Python 2. Thanks @ghisvail for the report (#101).

Internal

- Some tests in `pytest-mock`'s suite are skipped if assertion rewriting is disabled (#102).

1.6.3

- Fix `UnicodeDecodeError` during assert introspection in `assert_called_with` in Python 2. Thanks @AndreasHogstrom for the report (#91).

1.6.2

- Provide source package in `tar.gz` format and remove obsolete `MANIFEST.in`.

1.6.1

- Fix `mocker.resetall()` by ignoring mocker objects which don't have a `resetall` method, like for example `patch.dict`. Thanks @jdavisp3 for the PR (#88).

1.6.0

- The original assertions raised by the various `Mock.assert_*` methods now appear in the failure message, in addition to the message obtained from pytest introspection. Thanks @quodlibetor for the initial patch (#79).

1.5.0

- New `mocker.mock_module` variable points to the underlying mock module being used (`unittest.mock` or `mock`). Thanks @blueyed for the request (#71).

1.4.0

- New configuration variable, `mock_use_standalone_module` (defaults to `False`). This forces the plugin to import `mock` instead of `unittest.mock` on Python 3. This is useful to import a newer version than the one available in the Python distribution.
- Previously the plugin would first try to import `mock` and fallback to `unittest.mock` in case of an `ImportError`, but this behavior has been removed because it could hide hard to debug import errors (#68).
- Now `mock` (Python 2) and `unittest.mock` (Python 3) are lazy-loaded to make it possible to implement the new `mock_use_standlone_module` configuration option. As a consequence of this the undocumented `pytest_mock.mock_module` variable, which pointed to the actual mock module being used by the plugin, has been removed.
- DEFAULT is now available from the `mocker` fixture.

1.3.0

- Add support for Python 3.6. Thanks @hackebrot for the report (#59).
- `mock.mock_open` is now aliased as `mocker.mock_open` for convenience. Thanks @pokidovea for the PR (#66).

1.2

- Try to import `mock` first instead of `unittest.mock`. This gives the user flexibility to install a newer `mock` version from PyPI instead of using the one available in the Python distribution. Thanks @wcooley for the PR (#54).
- `mock.sentinel` is now aliased as `mocker.sentinel` for convenience. Thanks @kjwilcox for the PR (#56).

1.1

- From this version onward, `pytest-mock` is licensed under the MIT license (#45).
- Now the plugin also adds introspection information on differing call arguments when calling helper methods such as `assert_called_once_with`. The extra introspection information is similar to pytest's and can be disabled with the `mock_traceback_monkeypatch` option. Thanks @asfaltboy for the PR (#36).
- `mocker.stub()` now allows passing in the name for the constructed `Mock` object instead of having to set it using the internal `_mock_name` attribute directly. This is useful for debugging as the name is used in the mock's `repr` string as well as related assertion failure messages. Thanks @jurko-gospodnetic for the PR (#40).
- Monkey patching `mock` module for friendlier tracebacks is automatically disabled with the `--tb=native` option. The underlying mechanism used to suppress traceback entries from `mock` module does not work with that option anyway plus it generates confusing messages on Python 3.5 due to exception chaining (#44). Thanks @blueyed for the report.
- `mock.call` is now aliased as `mocker.call` for convenience. Thanks @jhermann for the PR (#49).

1.0

- Fix AttributeError with `mocker.spy` when spying on inherited methods (#42). Thanks @blueyed for the PR.

0.11.0

- PropertyMock is now accessible from `mocker`. Thanks @satyrius for the PR (#32).
- Fix regression using one of the `assert_*` methods in patched functions which receive a parameter named `method`. Thanks @sagarchalise for the report (#31).

0.10.1

- Fix regression in frozen tests due to `distutils` import dependency. Thanks @The-Compiler for the report (#29).
- Fix regression when using `pytest-mock` with `pytest-2.7.X`. Thanks @akscram for the report (#28).

0.10

- `pytest-mock` now monkeypatches the `mock` library to improve pytest output for failures of mock call assertions like `Mock.assert_called_with()`. Thanks to @Chronial for idea and PR (#26, #27)!

0.9.0

- New `mocker.resetall` function, which calls `reset_mock()` in all mocked objects up to that point. Thanks to @mathrick for the PR!

0.8.1

- `pytest-mock` is now also available as a wheel. Thanks @rouge8 for the PR!

0.8.0

- `mock.ANY` is now accessible from the mocker fixture (#17), thanks @tigarmo for the PR!

0.7.0

Thanks to @fogo, mocker.spy can now prey upon staticmethods and classmethods. :smile:

0.6.0

- Two new auxiliary methods, `spy` and `stub`. See `README` for usage. (Thanks @fogo for complete PR!)

0.5.0

- `Mock` and `MagicMock` are now accessible from the `mocker` fixture, many thanks to @marcwebbie for the complete PR!

0.4.3

- `mocker` fixture now returns the same object (#8). Many thanks to @RonnyPfannschmidt for the PR!

0.4.2

- Small fix, no longer using wheel as an alternate package since it conditionally depends on mock module based on Python version, as Python >= 3.3 already includes `unittest.mock`. Many thanks to @The-Compiler for letting me know and providing a PR with the fix!

0.4.1

- Small release that just uses `pytest_mock` as the name of the plugin, instead of `pytest-mock`: this makes it simple to depend on this plugin explicitly using `pytest_plugins` module variable mechanism.

0.4.0

- Changed fixture name from `mock` into `mocker` because it conflicted with the actual mock module, which made using it awkward when access to both the module and the fixture were required within a test.

  Thanks @kmosher for request and discussion in #4. :smile:

0.3.0

- Fixed bug #2, where a patch would not be uninstalled correctly after patching the same object twice.

0.2.0

- Added `patch.dict` support.

0.1.0

First release.